Name

intro – introduction to the X/Open Curses Package, which optimizes terminal screen handling and updating

Syntax

#include <cursesX.h>
cc [*options*] *files* **–lcursesX** [*libraries*]

Description

The `curses` (cursor optimization) package is the X/Open set of library routines used for writing screen-management programs. Cursor optimization minimizes the amount the cursor has to be moved around the screen in order to update it. Screen-management programs are used for tasks such as moving the cursor, printing a menu, dividing a terminal screen into windows or drawing a display on a screen for data entry and retrieval.

The `curses` package is split into three parts: screen updating, screen updating with user input, and cursor motion optimization. Screen-updating routines are used when parts of the screen need to be changed but the overall image remains the same. The cursor motion part of the package can be used separately for tasks such as defining how the cursor moves in response to tabs and newline characters

The `curses` routines do not write directly to the terminal screen (the physical screen): instead, they write to a window, a two-dimensional array of characters which represents all or part of the terminal screen. A window can be as big as the terminal screen or any smaller size down to a single character.

The `<cursesX.h>` header file supplies two default windows, `stdscr` (standard screen) and `curscr` (current screen) for all programs using `curses` routines. The `stdscr` window is the size of the current terminal screen. The `curscr` window is not normally accessed directly by the screen-management program; changes are made to the appropriate window and then the `refresh` routine is called. The screen program keeps track of what is on the physical screen and what is on `stdscr`. When `refresh` is called, it compares the two screen images and then sends a stream of characters to the terminal to make the physical screen look like `stdscr`.

The header file `<cursesX.h>` defines `stdscr` to be of the type `WINDOW*`. This is a pointer to a C structure which includes the starting position of the window on the screen and the window size.

Some `curses` routines are designed to work with a `pad`. A pad is a type of window whose size is not restricted by the size of the screen. Use a pad when you only need part of a window on the screen at any one time, for example when running a spreadsheet application.

Other windows can be created with `newwin` and used instead of `stdscr` for maintaining several different screen images, for example, one window can control input/output and another can display error messages. The routine `subwin` creates subwindows within windows. When windows overlap, the contents of the current screen show the most recently refreshed window.

Among the most basic routines are move and addch. These routines are used to move the cursor around and to add characters to the default window, stdscr.

All curses data is manipulated using the routines provided by the curses library. You should not use routines or system calls from other libraries in a curses program as they may cause undesirable results when you run the program.

Using Curses

The curses library has three types of routines; Main routines, TERMINFO routines and TERMCAP compatibility routines

The terminfo routines are a group of routines within the curses library which provide a database containing descriptions of many terminals that can be used with curses programs. The termcap compatibility routines are provided as a conversion aid for programs using termcap.

Most screen handling can be achieved using the Main routines. The following hints should help you make the most of the screen-handling routines.

The <cursesX.h> header file must always be included whenever curses functions are used in a program. Note that the header file includes <sgtty.h> to enable the terminal to use the features provided by ULTRIX. All the manual definitions assume that <cursesX.h> has been included in the code.

The header file defines global variables and data structures, and defines several of the routines as macros. The integer variables LINES and COLS are defined so that when a curses program is run on a particular terminal, initscr assigns the vertical and horizontal dimensions of the terminal screen to these variables.

A curses program must start by calling the routine initscr to allocate memory space for the windows. It should only be called once in a program, as it can overflow core memory if it is called repeatedly. The routine endwin is used to exit from the screen-handling routines.

Most interactive screen-oriented programs need character-at-a-time input without echoing. To achieve this, you should call:

```
nonl();
cbreak();
noecho();
```

immediately after calling initscr. All curses routines that move the cursor, move it relative to the home position in the upper left corner of the screen. The (LINES, COLS) coordinate at this position is (1,1). Note that the vertical coordinate y is given first and the horizontal coordinate x is given second. The -1 in the example program takes the home position into account to place the cursor on the centre line of the terminal screen. The example program displays **MIDSCREEN** in the centre of the screen. Use the refresh routine after changing a screen to make the terminal screen look like stdscr.

Example Program

```
#include <cursesX.h>
main ()
{

initscr();      /*initialize terminal settings, data
                ** structures and variables*/
move(LINES/2 -1, COLS/2 -4);
```

```
addstr("MID");
refresh();      /* send output to update terminal
                 **  screen */
addstr("SCREEN");
refresh();      /* send more output to terminal
                 **  screen */
endwin();       /*restore all terminal settings */

}
```

Main Routines

Routines listed here can be called when using the `curses` library. Routines that are preceded by a **w** affect a specified window, those preceded by a **p** affect a specified pad. All other routines affect the default window `stdscr`. Windows are specified by a numeric argument, for example: `winch` (*win*) where *win* is the specified window.

addch(ch)	Add a character to stdscr (like putchar wraps to next line at end of line)
addstr(str)	Call addch with each character in *str*
attroff(attrs)	Turn off named attributes
attron(attrs)	Turn on named attributes
attrset(attrs)	Set current attributes to *attrs*
baudrate( )	Display current terminal speed
beep( )	Sound beep on terminal
box(win, vert, hor)	Draw a box around edges of *win*, *vert* and *hor* are characters to use for vertical and horizontal edges of box
clear( )	Clear stdscr
clearok(win, bf)	Clear screen before next redraw of *win*
clrtobot( )	Clear to bottom of stdscr
clrtoeol( )	Clear to end of line on stdscr
cbreak( )	Set cbreak mode
delay_output(ms)	Insert *ms* millisecond pause in output
delch( )	Delete a character
deleteln( )	Delete a line
delwin(win)	Delete *win*
doupdate( )	Update screen from all wnoutrefresh
echo( )	Set echo mode
endwin( )	End window modes
erase( )	Erase stdscr
erasechar( )	Return user's erase character
fixterm( )	Restore tty to in ''curses'' state
flash( )	Flash screen or beep
flushinp( )	Throw away any typeahead
getch( )	Get a character from tty
getstr(str)	Get a string through stdscr
gettmode( )	Establish current tty modes
getyx(win, y, x)	Get (y, x) coordinates
has_ic( )	True if terminal can do insert character
has_il( )	True if terminal can do insert line
idlok(win, bf)	Use terminal's insert/delete line if bf != 0

inch( )	Get character at current (y, x) coordinates
initscr( )	Initialize screens
insch(c)	Insert a character
insertln( )	Insert a line
intrflush(win, bf)	Interrupt flush output if bf is TRUE
keypad(win, bf)	Enable keypad input
killchar( )	Return current user's kill character
leaveok(win, flag)	Leave cursor anywhere after refresh if flag!=0 for *win*. Otherwise cursor must be left at current position
longname( )	Return verbose name of terminal
meta(win, flag)	Allow meta characters on input if flag != 0
move(y, x)	Move to (y, x) on stdscr

NOTE: The following routines prefixed with **mv** require `y` and `x` coordinates to move to, before performing the same functions as the standard routines. As an example, `mvaddch` performs the same function as `addch`, but `y` and `x` coordinates must be supplied first. The routines prefixed with **mvw** also require a window or pad argument.

mvaddch(y, x, ch)	
mvaddstr(y, x, str)	
mvcur(oldrow, oldcol, newrow, newcol)	low level cursor motion
mvdelch(y, x)	
mvgetch(y, x)	
mvgetstr(y, x)	
mvinch(y, x)	
mvinsch(y, x, c)	
mvprintw(y, x, fmt, args)	
mvscanw(y, x, fmt, args)	
mvwaddch(win, y, x, ch)	
mvwaddstr(win, y, x, str)	
mvwdelch(win, y, x)	
mvwgetch(win, y, x)	
mvwgetstr(win, y, x)	
mvwin(win, by, bx)	
mvwinch(win, y, x)	
mvwinsch(win, y, x, c)	
mvwprintw(win, y, x, fmt, args)	
mvwscanw(win, y, x, fmt, args)	
newpad(nlines, ncols)	Create a new pad with given dimensions
newterm(type, fd)	Set up new terminal of given type to output on fd
newwin(lines, cols, begin_y, begin_x)	Create a new window
nl( )	Set newline mapping
nocbreak( )	Unset cbreak mode
nodelay(win, bf)	Enable nodelay input mode through getch
noecho( )	Unset echo mode
nonl( )	Unset newline mapping
noraw( )	Unset raw mode

overlay(win1, win2)	Overlay win1 on win2
overwrite(win1, win2)	Overwrite win1 on top of win2
pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)	Like prefresh but with no output until doupdate called
prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)	Refresh from pad starting with given upper left corner of pad with output to given portion of screen
printw(fmt, arg1, arg2, ...)	printf on stdscr
raw( )	Set raw mode
refresh( )	Make current screen look like stdscr
resetterm( )	Set tty modes to ''out of curses'' state
resetty( )	Reset tty flags to stored value
saveterm( )	Save current modes as ''in curses'' state
savetty( )	Store current tty flags
scanw(fmt, arg1, arg2, ...)	scanf through stdscr
scroll(win)	Scroll *win* one line
scrollok(win, flag)	Allow terminal to scroll if flag != 0
set_term(new)	Switch between different terminals
setscrreg(t, b)	Set user scrolling region to lines t through b
setupterm(term, filenum, errret)	Low level terminal setup
standend( )	Clear standout mode attribute
standout( )	Set standout mode attribute
subwin(win, lines, cols, begin_y, begin_x)	Create a subwindow
touchwin(win)	''change'' all of *win*
traceoff( )	Turn off debugging trace output
traceon( )	Turn on debugging trace output
typeahead(fd)	Use file descriptor fd to check typeahead
unctrl(ch)	Produce printable version of *ch*
waddch(win, ch)	Add character to *win*
waddstr(win, str)	Add string to *win*
wattroff(win, attrs)	Turn off attrs in *win*
wattron(win, attrs)	Turn on attrs in *win*
wattrset(win, attrs)	Set attrs in *win* to attrs
wclear(win)	Clear *win*
wclrtobot(win)	Clear to bottom of *win*
wclrtoeol(win)	Clear to end of line on *win*
wdelch(win, c)	Delete char from *win*
wdeleteln(win)	Delete line from *win*
werase(win)	Erase *win*
wgetch(win)	Get a character through *win*
wgetstr(win, str)	Get a string through *win*
winch(win)	Get character at current (y, x) in *win*
winsch(win, c)	Insert char into *win*
winsertln(win)	Insert line into *win*
wmove(win, y, x)	Set current (y, x) coordinates on *win*
wnoutrefresh(win)	Refresh but no screen output
wprintw(win, fmt, arg1, arg2, ...)	printf on *win*
wrefresh(win)	Make screen look like *win*
wscanw(win, fmt,	scanf through *win*

arg1, arg2, ...)	
wsetscrreg(win, t, b)	Set scrolling region of *win*
wstandend(win)	Clear standout attribute in *win*
wstandout(win)	Set standout attribute in *win*

Caution

The plotting library `plot`(3x) and the `curses`(3cur) library both use the names `erase()` and `move()`. The `curses` versions are macros. If you need both libraries, put the `plot`(3x) code in a different source file to the `curses`(3cur) code, and/or `#undef move()` and `erase()` in the `plot`(3x) code.

TERMINFO Level Routines

If the environment variable TERMINFO is defined, any program using `curses` will check for a local terminal definition before checking in the standard libraries. For example, if the standard place is `/usr/lib/terminfo`, and set to **vt100**, the compiled file will normally be found in `/usr/lib/terminfo/v/vt100`. The **v** is copied from the first letter of vt100 to avoid creating huge directories. However, if TERMINFO is set to `/usr/mark/myterms`, `curses` will first check `/usr/mark/myterms/v/vt100`, and if that fails, will then check `/usr/lib/terminfo/v/vt100`. This is useful for developing experimental definitions or when there is no write permission for `/usr/lib/terminfo`.

These routines should be called by programs that need to deal directly with the `terminfo` database, but as this is a low level interface, it is not recommended.

Initially, the routine `setupterm` should be called. This will define the set of terminal-dependent variables defined in `terminfo`(5). The include files `<cursesX.h>` and `<term.h>` should be included to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through `tparm` to instantiate them. All terminfo strings (including the output of `tparm`) should be printed with `tputs` or `putp`. Before exiting, `resetterm` should be called to restore the tty modes.

Programs which want shell escapes or <CTRL/Z> suspending can call `resetterm` before the shell is called and `fixterm` after returning from the shell.

fixterm( )	Restore tty modes for terminfo use (called by setupterm)
resetterm( )	Reset tty modes to state before program entry
setupterm(term, fd, rc)	Read in database. Terminal type is the character string term, all output is to ULTRIX System file descriptor fd. A status value is returned in the integer pointed to by rc: 1 is normal. The simplest call would be setupterm(0, 1, 0) which uses all defaults
tparm(str, p1, p2, ..., p9)	Instantiate string str with parms p_i
tputs(str, affcnt, putc)	Apply padding info to string str affcnt is the number of lines affected, or 1 if not applicable. Putc is a putchar-like function to which the characters are passed, one at a time

putp(str)	A function that calls tputs (str, 1, putchar)
vidputs(attrs, putc)	Output the string to put terminal in video attribute mode attrs, which is any combination of the attributes listed below Chars are passed to putchar-like function putc
vidattr(attrs)	Like vidputs but outputs through putchar

Termcap Compatibility Routines

The following routines were included as a conversion aid for programs that use termcap. Their parameters are the same as for termcap. They are emulated using the `terminfo` database.

DO NOT use these routines in new programs.

tgetent(bp, name)	Look up termcap entry for name
tgetflag(id)	Get boolean entry for id
tgetnum(id)	Get numeric entry for id
tgetstr(id, area)	Get string entry for id
tgoto(cap, col, row)	Apply parms to given cap
tputs(cap, affcnt, fn)	Apply padding to cap calling fn as putchar

As an aid to compatibility, the object module `termcap.o` has been provided in `/usr/lib/termcap.o`. This module should be linked into an application before resolving against the `curses` library. If your application contains references such as `UP` then recompile using

```
cc [options] files /usr/lib/termcap.o -lcursesX[libs]
```

Errors

No errors are defined for the `curses` functions.

Return Values

For most `curses` routines, the `OK` value is returned if a routine is properly completed and the `ERR` value is returned if some error occurs.

See Also

ioctl(2), getenv(3), printf(3s), putchar(3s), scanf(3s), plot(3x), terminfo(5), tic(1), termcap(5)
Guide to X/Open Curses Screen-Handling

addch (3cur)

Name

addch, waddch, mvaddch, mvwaddch – add character to window

Syntax

#include <cursesX.h>

int addch(ch)
chtype ch;

int waddch(win, ch)
WINDOW *win;
chtype ch;

int mvaddch(y, x, ch)
int y, x;
chtype ch;

int mvwaddch(win, y, x, ch)
WINDOW *win;
int y, x;
chtype ch;

Description

The routine `addch` inserts the character `ch` into the default window at the current cursor position and the window cursor is advanced. The character is of the type `chtype` which is defined in the `<cursesX.h>` header file, as containing both data and attributes.

The routine `waddch` inserts the character `ch` into the specified window at the current cursor position. The cursor position is advanced.

The routine `mvaddch` moves the cursor to the specified (y, x) position and inserts the character `ch` into the default window. The cursor position is advanced after the character has been inserted.

The routine `mvwaddch` moves the cursor to the specified (y, x) position and inserts the character `ch` into the specified window. The cursor position is advanced after the character has been inserted.

All these routines are similar to `putchar`. The following information applies to all the routines.

If the cursor moves on to the right margin, an automatic newline is performed. If `scrollok` is enabled, and a character is added to the bottom right corner of the screen, the scrolling region will be scrolled up one line. If scrolling is not allowed, ERR will be returned.

If `ch` is a tab, newline, or backspace, the cursor will be moved appropriately within the window. If `ch` is a newline, the `clrtoeol` routine is called before the cursor is moved to the beginning of the next line. If newline mapping is off, the cursor will be moved to the next line, but the x coordinate will be unchanged. If `ch` is a tab the cursor is moved to the next tab position within the window. If `ch` is another control character, it will be drawn in the ^X notation. Calling the `inch` routine after adding a control character returns the representation of the control character, not the control character.

Video attributes can be combined with a character by or-ing them into the parameter. This will result in these attributes being set. The intent here is that text, including attributes, can be copied from one place to another using `inch` and `addch`. For further information, see `standout`(3cur).

The `addch`, `mvaddch`, and `mvwaddch` routines are macros.

Return Value

The `addch`, `waddch`, `mvaddch`, and `mvwaddch` functions return OK on success and ERR on error.

See Also

clrtoeol(3cur), inch(3cur), scrollok(3cur), standout(3cur), putchar(3s)

Name

addstr, waddstr, mvaddstr, mvwaddstr – add string to window

Syntax

```
#include <cursesX.h>

int addstr(str)
char *str;

int waddstr(win, str)
WINDOW *win;
char *str;

int mvaddstr(y, x, str)
int y, x;
char *str;

int mvwaddstr(win, y, x, str)
WINDOW *win;
int y, x;
char *str;
```

Description

The `addstr` routine writes all the characters of the null-terminated character string `str` on the default window at the current (y, x) coordinates.

The routine `waddstr` writes all the characters of the null terminated character string `str` on the specified window at the current (y, x) coordinates.

The routine `mvaddstr` writes all the characters of the null terminated character string `str` on the default window at the specified (y, x) coordinates.

The routine `mvwaddstr` writes all the characters of the null terminated character string `str` on the specified window at the specified (y, x) coordinates.

The following information applies to all the routines. All the routines return `ERR` if writing the string causes illegal scrolling. In this case the routine will write as much as possible of the string on the window.

These routines are functionally equivalent to calling `addch` or `waddch` once for each character in the string.

The routines `addstr`, `mvaddstr`, and `mvwaddstr` are macros.

Return Value

The `addstr`, `waddstr`, `mvaddstr`, and `mvwaddstr` functions return OK on success and ERR on error.

See Also

addch(3cur), waddch(3cur)

Name

attroff, attron, attrset, standend, standout, wstandend, wstandout, wattroff, wattron, wattrset – attribute manipulation

Syntax

```
#include <cursesX.h>

int attroff(attrs)
int attrs;

int wattroff(win, attrs)
WINDOW *win;
int attrs;

int attron(attrs)
int attrs;

int wattron(win, attrs)
WINDOW *win;
int attrs;

int attrset(attrs)
int attrs;

int wattrset(win, attrs)
WINDOW *win;
int attrs;

int standend( )

wstandend(win)
WINDOW *win;

int standout( )

int wstandout(win)
WINDOW *win;
```

Description

These routines manipulate the current attributes of a window.

The routine `attroff` turns off the named attributes (`attrs`) of the default window without turning any other attributes on or off.

The routine `attron` turns on the named attributes of the default window without affecting any other attributes.

The routine `attrset` sets the current attributes of the default window to the named attributes `attrs`, which is of the type `chtype`, and is defined in the `<cursesX.h>` header file.

The routine `standout` switches on the best highlighting mode available on the terminal for the default window and it is functionally the same as `attron(A_STANDOUT1)`.

The routine `standend` switches off all highlighting associated with the default window. It is functionally the same as `attrset`(0), in that it turns off all attributes.

The routine `wattroff` switches off the named attributes, `attrs`, for the specified window. Other attributes are not changed.

The routine `wattron` turns on the named attributes of the specified window without affecting any others.

The routine `wattrset` sets the current attributes of the specified window to `attrs`.

The routine `wstandout` switches on the best highlighting mode available on the terminal for the specified window. Functionally it is the same as `wattron(A_STANDOUT1)`.

The routine `wstandend` switches off all highlighting associated with the specified window. Functionally it is the same as `wattrset`(0); that is, it turns off all attributes.

Attributes

Attributes can be any combination of A_STANDOUT, A_REVERSE, A_BOLD, A_DIM, A_BLINK and A_UNDERLINE. These constants are defined in the `<cursesX.h>` header file. They are also described in the *Guide to X/Open Curses Screen-Handling*. Attributes can be combined with the C language | `(or)` operator.

The current attributes of a window are applied to all characters that are written into the window with `addch` or `waddch`. Attributes are properties of the character, and move with the character through any scrolling and insert/delete line/character operations. Within the restrictions set by the terminal hardware they will be displayed as the graphic rendition of characters put on the screen.

The routines `attroff`, `attron` and `attrset` are macros.

Return Value

The `attroff`, `wattroff`, `attron`, `wattron`, `attrset`, `wattrset`, `standend`, `wstandend`, `standout`, and `wstandout` functions return OK on success and ERR on error.

See Also

addch(3cur)
Guide to X/Open Curses Screen-Handling

Name

baudrate – return terminal baudrate

Syntax

int baudrate( )

Description

The `baudrate` routine returns the output speed of the terminal in bits per second, for example 9600, as an integer.

Return Value

The `baudrate` function returns the baudrate in bits per second.

Name

beep, flash – generate audiovisual alarm

Syntax

#include <cursesX.h>

int beep( )

int flash( )

Description

The `beep` routine sounds the audible alarm on the terminal, if possible, otherwise it flashes the screen.

The routine `flash` flashes the screen, if possible, otherwise it sounds the audible alarm.

If neither signal can be used on a particular terminal, nothing happens.

Return Value

The `beep` and `flash` functions return OK on success and ERR on error.

Name

box – draw box

Syntax

#include <cursesX.h>

int box(win, vert, hor)
WINDOW *win;
chtype vert, hor;

Description

The `box` routine draws a box around the edge of the window. The arguments `vert` and `hor` are the vertical and horizontal characters the box is to be drawn with.

If `vert` and `hor` are 0 or unspecified, then default characters are used.

If scrolling is disabled and the window encompasses the bottom right corner of the screen, all corners are left blank to avoid an illegal scroll.

Return Value

The `box` function returns OK on success and ERR on error.

Name

cbreak, nocbreak – set/clear cbreak mode

Syntax

int cbreak( )

int nocbreak( )

Description

The routine `cbreak` puts the terminal into CBREAK mode. In this mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. Interrupt and flow control characters are unaffected by this mode.

The routine `nocbreak` disables CBREAK. In this case the terminal driver will buffer input until a newline or carriage return is typed.

The initial settings that determine whether or not a terminal is in CBREAK mode are dependent on the terminal driver implementation. As a result of this, it is not possible to determine if a terminal is in CBREAK mode, as it is an inherited characteristic. It is necessary to call `cbreak` to ensure that the terminal is set to the correct mode for the application.

Return Value

The `cbreak` and `nobreak` functions return OK on success and ERR on error.

Name

clear, wclear – clear window

Syntax

#include <cursesX.h>

int clear( )

int wclear(win)
WINDOW *win;

Description

The `clear` routine resets the entire default window to blanks and sets the current (y, x) coordinates to (0, 0).

The routine `wclear` resets the entire specified window to blanks and sets the current (y, x) coordinates to (0, 0).

The `clear` routine assumes that the screen may have garbage on it that it doesn't know about. The routine first calls `erase` which copies blanks to every position in the default window, and then `clearok`, which clears the physical screen completely on the next call to `refresh` for `stdscr`.

The routine `clear` is a macro.

Return Value

The `clear` and `wclear` functions return OK on success and ERR on error.

See Also

clearok(3cur), erase(3cur), refresh(3cur)

Name

clearok – enable screen clearing

Syntax

#include <cursesX.h>

int clearok(win, bf)
WINDOW *win;
bool bf;

Description

If `bf` is `TRUE`, the next call to `refresh`(3cur) for the specified window will clear the window completely and redraw the entire window without changing the original screen's contents. This is useful when the contents of the screen are uncertain. If the window is `stdscr` the entire screen is redrawn.

Return Value

The `clearok` function returns OK on success and ERR on error.

See Also

refresh(3cur)

Name

clrtobot, wclrtobot – clear to end of screen

Syntax

#include <cursesX.h>

int clrtobot( )

int wclrtobot(win)
WINDOW *win;

Description

The `clrtobot` routine begins at the current cursor position in the default window and changes the remainder of the screen to blanks. The current cursor position is also changed to a blank.

The `wclrtobot` routine begins at the current cursor position in the specified window and changes the rest of the screen to blanks, including the current cursor position.

The routine `clrtobot` is a macro.

Return Value

The `clrtobot` and `wclrtobot` functions return OK on success and ERR on error.

Name

clrtoeol, wclrtoeol – clear to end of line

Syntax

#include <cursesX.h>

int clrtoeol( )

int wclrtoeol(win)
WINDOW *win;

Description

The `clrtoeol` routine erases the current line to the right of the cursor, inclusive, on the default window.

The routine `wclrtoeol` erases the current line to the right of the cursor, inclusive, on the specified window.

The routine `clrtoeol` is a macro.

Return Value

The `clrtoeol` and `wclrtoeol` functions return OK on success and ERR on error.

Name

def_prog_mode, def_shell_mode – save terminal modes

Syntax

int def_prog_mode( )

int def_shell_mode( )

Description

The `def_prog_mode` routine saves the current terminal modes as the **program** if the terminal is running under `curses`. The stored terminal modes are used by the `reset_prog_mode`(3cur) routine. This function is used when the user makes a temporary exit from `curses`.

The routine `def_shell_mode` saves the current terminal modes as the **shell** if the terminal is not running under `curses`. The stored terminal modes are used by the `reset_shell_mode`(3cur) routine.

Both routines are called automatically by `initscr`(3cur).

Return Value

The `def_prog_mode` and `def_shell_mode` functions return OK on success and ERR on error.

See Also

initscr(3cur), reset_prog_mode(3cur), reset_shell_mode(3cur)

delay_output(3cur)

Name

delay_output – cause short delay

Syntax

int delay_output(ms)
int ms;

Description

Insert 10 x `ms` millisecond pause in output. The largest number allowed for `ms` is 0.5 seconds (500 milleseconds).

Return Value

The `delay_output` function returns OK on success and ERR on error.

Name

delch, mvdelch, mvwdelch, wdelch – remove character from window

Syntax

```
#include <cursesX.h>

int delch( )

int wdelch(win)
WINDOW *win;

int mvdelch(y, x)
int y, x;

int mvwdelch(win, y, x)
WINDOW *win;
int y, x;
```

Description

The `delch` routine deletes the character under the cursor in the default window. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change.

The routine `wdelch` deletes the character under the cursor in the specified window. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change.

The routine `mvdelch` moves the cursor to the specified position in the default window. The character found at this location is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change.

The routine `mvwdelch` moves the cursor to the specified position in the specified window. The character found at this location is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change.

The routines `delch`, `mvdelch` and `mvwdelch` are macros.

Return Value

The `delch`, `mvdelch`, `mvwdelch` and `wdelch` functions return OK on success and ERR on error.

Name

deleteln, wdeleteln – remove line from window

Syntax

#include <cursesX.h>

int deleteln( )

int wdeleteln(win)
WINDOW *win;

Description

The `deleteln` routine deletes the current line of the default window. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change.

The routine `wdeleteln` deletes the current line of the specified window. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change.

The routine `deleteln` is a macro.

Return Value

The `deleteln` and `wdeleteln` functions return OK on success and ERR on error.

Name

delwin – delete window

Syntax

#include <cursesX.h>

int delwin(win)
WINDOW *win;

Description

The `delwin` routine deletes the named window, freeing all memory associated with it. Where windows overlap, subwindows should be deleted before the main window.

Return Value

The `delwin` function returns OK on success and ERR on error.

draino(3cur)

Name

draino – wait for output to drain

Syntax

draino(ms)
int ms;

Description

This function waits until there is only `ms` milliseconds worth of output left in the output queue. The restrictions on the number of milleseconds delay are determined by `napms`(3cur).

See Also

napms(3cur)

Name

echo, noecho – enable/disable terminal echo

Syntax

int echo( )

int noecho( )

Description

The `echo` routine enables echoing of characters typed by the user. The `noecho` routine disables echoing of characters typed by the user.

Initially, input characters are echoed. Subsequent calls to `echo` and `noecho` do not flush typeahead.

Return Value

The `echo` and `noecho` functions return OK on success and ERR on error.

endwin(3cur)

Name

endwin – restore initial terminal environment

Syntax

int endwin( )

Description

This routine restores tty modes, moves the cursor to the lower left corner of the screen and resets the terminal to the last non-`curses` mode.

A program should always call `endwin` before exiting or escaping from `curses` mode temporarily. Call `refresh` or `doupdate` to resume after a temporary escape.

Return Value

The `endwin` function returns OK on success and ERR on error.

See Also

doupdate(3cur), refresh(3cur)

Name

erase, werase – copy blanks into window

Syntax

#include <cursesX.h>

int erase( )

int werase(win)
WINDOW *win;

Description

The `erase` routine copies blanks to every position in the default window, the `werase` routine copies blanks to every position in the specified window.

The routine `erase` is a macro.

Return Value

The `erase` and `werase` functions return OK on success and ERR on error.

Name

erasechar – return current ERASE character

Syntax

#include <cursesX.h>

char erasechar( )

Description

The user's current erase character is returned.

Return Value

The `erasechar` function returns the user's current erase character.

Name

flushinp – discard typeahead

Syntax

#include <cursesX.h>

int flushinp( )

Description

Any typeahead input that has not been read by the program is discarded.

Return Value

The `flushinp` function returns OK on success and ERR on error.

See Also

typeahead(3cur)

Name

getch, mvgetch, mvwgetch, wgetch – read character

Syntax

```
#include <cursesX.h>

int getch( )

int wgetch(win)
WINDOW *win;

int mvgetch(y, x)
int y, x;

int mvwgetch(win, y, x)
WINDOW *win;
int y, x;
```

Description

The `getch` routine reads a character from the terminal associated with the default window.

The `wgetch` routine reads a character from the terminal associated with the specified window.

The routine `mvgetch` reads a character from the terminal associated with the default window at the specified position.

The routine `mvwgetch` reads a character from the terminal associated with the specified window at the specified position.

The following information applies to all the routines. In `nodelay` mode, if there is no input waiting, the integer `ERR` is returned. In `delay` mode, the program waits until the system passes text through to the program. Usually the program will restart after one character or after the first newline, but this depends on how `cbreak` is set. The character will be echoed on the designated window unless `noecho` has been set.

If `keypad` is `TRUE`, and a function key is pressed, the token for that function key is returned instead of the raw characters. Possible function keys are defined in the `<cursesX.h>` header file with integers beginning with 0401. The function key names begin with KEY_. Function keys and their respective integer values are described in the *Guide to X/Open Curses Screen-Handling*

If a character is received that could be the beginning of a function key (such as escape), `curses` sets a timer. If the remainder of the sequence does not come within the designated time, the character will be passed through, otherwise the function key value is returned. Consequently, there may be a delay after a user presses the escape key before the escape is returned to the program.

Using the escape key for a single character function is discouraged.

The routines `getch`, `mvgetch` and `mvwgetch` are macros.

Return Value

Upon successful completion, the `getch`, `mvgetch`, and `wgetch` functions return the character read.

If in delay mode and no data is available, ERR is returned.

See Also

cbreak(3cur), keypad(3cur), nodelay(3cur), noecho(3cur)
Guide to X/Open Curses Screen-Handling

Name

getstr, mvgetstr, mvwgetstr, wgetstr – read string

Syntax

```
#include <cursesX.h>

int getstr(str)
char *str;

int wgetstr(win, str)
WINDOW *win;
char *str;

int mvgetstr(y, x, str)
int y, x;
char *str;

int mvwgetstr(win, y, x, str)
WINDOW *win;
int y, x;
char *str;
```

Description

The `getstr` routine reads characters from the terminal associated with the default window and stores them in a buffer until a carriage return or newline is received from `stdscr`. The routine `getch` B is called by `getstr` to read each character.

The routine `wgetstr` reads characters from the terminal associated with the specified window. The characters are read from the current cursor position until a newline or carriage return is received.

The routine `mvgetstr` reads characters from the terminal associated with the default window. The characters are read from the specified cursor position until a newline or carriage return is received.

The routine `mvwgetstr` reads characters from the terminal associated with the specified window. The characters are read from the specified cursor position until a newline or carriage return is received.

The following information applies to all the routines.

The resulting string is placed in the area pointed to by the character pointer `str`. The user's erase and kill characters are interpreted. The area used to hold the string is assumed to be large enough to handle it, as `getstr` does not check for buffer overflow. If the area is not large enough, the result will be unpredictable.

The routines `getstr`, `mvgetstr` and `mvwgetstr` are macros.

Return Value

The `getstr`, `mvgetstr`, `mvwgetstr` and `wgetstr` functions return OK on success and ERR on error.

See Also

getch(3cur)

getyx (3cur)

Name

getyx – get cursor position

Syntax

#include <cursesX.h>

int getyx(win, y, x)
WINDOW *win;
int y, x;

Description

The cursor coordinates of the window are placed in the two integer variables `y` and `x`. This routine is implemented as a macro, so no `&` is necessary before the variables.

Return Value

No return value is defined for this function.

Name

has_ic – determine whether insert/delete character available

Syntax

#include <cursesX.h>

bool has_ic( )

Description

True if the terminal has insert- and delete-character capabilities.

The routines `insch` and `delch` are always available in the `curses` library if the terminal does not have the required capabilities.

Return Value

This function returns TRUE if the terminal has insert character and delete character capabilities, otherwise it returns FALSE.

See Also

delch(3cur), insch(3cur)

Name

has_il – determine whether insert/delete line is available

Syntax

#include <cursesX.h>

bool has_il( )

Description

This function will return the value `TRUE` if the terminal has insert- and delete-line capabilities, or if it can simulate them using scrolling regions. This function might be used to check if it would be appropriate to turn on physical scrolling using the `scrollok` routine.

The routines `insertln` and `deleteln` are always available in the `curses` library if the terminal does not have the required facilities.

Return Value

This function returns TRUE if the terminal has insert line and delete line capabilities, or can simulate them using scrolling regions, otherwise it returns FALSE.

See Also

deleteln(3cur), insertln(3cur), scrollok(3cur)

Name

idlok – enable use of insert/delete line

Syntax

#include <cursesX.h>

int idlok(win, bf)
WINDOW *win;
bool bf;

Description

If enabled (`bf` is `TRUE`), `curses` uses the insert/delete line hardware of terminals if it is available. If disabled, `curses` will not use this feature. This option should be enabled only if the application needs insert/delete line; for example, for a screen editor. It is disabled by default as insert/delete line can be visually annoying when used in some applications.

If insert/delete line cannot be used, `curses` will redraw the changed portions of all lines.

NOTE

The terminal hardware insert/delete character feature is always used if available.

Return Value

The `idlok` function returns OK on success and ERR on error.

Name

inch, mvinch, mvwinch, winch – return character from window

Syntax

#include <cursesX.h>

chtype inch()

chtype winch(win)
WINDOW *win;

chtype mvinch(y, x)
int y, x;

chtype mvwinch(win, y, x)
WINDOW *win;
int y, x;

Description

The `inch` routine returns the character at the current cursor position in the default window. If any attributes are set for that character, their values will be `or`-ed into the value returned.

The routine `mvinch` returns the character at the specified position in the default window. If any attributes are set for that position, their values will be `or`-ed into the value returned.

The `winch` routine returns the character at the current position in the named window. If any attributes are set for that position, their values will be `or`-ed into the value returned.

The `mvwinch` routine returns the character at the specified position in the named window. If any attributes are set for that position, their values will be `or`-ed into the value returned.

The following information applies to all the routines.

The predefined constants A_CHARTEXT and A_ATTRIBUTES, defined in `<cursesX.h>`, can be used with the `&` (logical `and`) operator to extract the character or attributes alone.

The `inch, winch, mvinch` and `mvwinch` routines are macros.

Return Value

Upon successful completion, the `inch`, `mvinch`, `mvwinch` and `winch` functions return the character at the selected position. Otherwise, the `mvinch` and `mvwinch` functions return ERR.

Name

initscr – initialize terminal environment

Syntax

#include <cursesX.h>

WINDOW *initscr

Description

This routine determines the terminal type, initializes all `curses` data structures and allocates memory space for the windows. It also arranges that the first call to the `refresh` routine will clear the screen.

The first routine called in a program using `curses` routines should almost always be `initscr`. If errors occur, `initscr` will write an appropriate error message to standard error and exit. If the program needs an indication of error conditions, `newterm` should be used instead of `initscr`.

Note that the `curses` program should only call `initscr` once as it may overflow core memory if it is called repeatedly. If this does occur, ERR is returned.

Return Value

The `initscr` function returns `stdscr` on success, and calls `exit` on error.

See Also

newterm(3cur), refresh(3cur)

Name

insch, mvinsch, mvwinsch, winsch – insert character

Syntax

```
#include <cursesX.h>

int insch(ch)
chtype ch;

int winsch(win, ch)
WINDOW *win;
chtype ch;

int mvinsch(y, x, ch)
int y, x;
chtype ch;

int mvwinsch(win, y, x, ch)
WINDOW *win;
int y, x;
chtype ch;
```

Description

The `insch` routine inserts the character `ch` at the current cursor position on the default window.

The `mvinsch` routine inserts the character `ch` at the specified cursor position on the default window.

The `winsch` routine inserts the character `ch` at the current cursor position on the specified window.

The `mvwinsch` routine inserts the character `ch` at the specified cursor position on the specified window.

All the routines cause the following actions. All characters from the cursor position to the right edge are moved one space to the right. The last character on the line is always lost, even if it is a blank. The cursor position does not change after the insert is completed.

The `insch`, `mvinsch` and `mvwinsch` routines are macros.

Return Value

The `insch`, `mvinsch`, `mvwinsch`, and `winsch` functions return OK on success and ERR on error.

Name

insertln, winsertln – insert line

Syntax

#include <cursesX.h>

int insertln( )

int winsertln(win)
WINDOW *win;

Description

The `insertln` routine inserts a blank line above the current line in the default window. All lines below and including the current line are moved down. The bottom line is lost and the current line becomes blank. The (y, x) coordinates are unchanged.

The `winsertln` routine inserts a blank line above the current line on the specified window. All lines below and including the current line are moved down. The bottom line is lost and the current line becomes blank. The (y, x) coordinates are unchanged.

The routine `insertln` is a macro.

Return Value

The `insertln` and `winsertln` functions return OK on success and ERR on error.

Name

intrflush – enable flush on interrupt

Syntax

```
#include <cursesX.h>

int intrflush(win, bf)
WINDOW *win;
bool bf;
```

Description

If `intrflush` is enabled, pressing an interrupt key (interrupt, break, quit) flushes all output in the tty driver queue. This gives the effect of a faster response to the interrupt but causes the `curses` program to have an inaccurate picture of what is on the screen. Disabling the option prevents the flush.

The default for the option is dependent on the tty driver settings. You have to force the terminal into the state you require. The window argument is ignored.

Return Value

The `intrflush` function returns OK on success and ERR on error.

Name

keypad – enable keypad

Syntax

#include <cursesX.h>

int keypad(win, bf)
WINDOW *win;
bool bf;

Description

This option enables the keypad of the user's terminal. If the keypad is enabled, pressing a function key (such as an arrow key) will return a single value representing the function key. For example, pressing the left arrow key results in the value KEY_LEFT being returned.. For more information see the *Guide to X/Open Curses Screen-Handling*.

The routine `getch` is used to return the character. If the keypad is disabled, `curses` does not treat function keys as special keys and the program interprets the escape sequences itself. Keypad layout is terminal dependent; some terminals do not even have a keypad.

Return Value

The `keypad` function returns OK on success and ERR on error.

See Also

getch(3cur)
Guide to X/Open Curses Screen-Handling

Name

killchar – return current kill character

Syntax

#include <cursesX.h>

char killchar( )

Description

The user's current line kill character is returned.

Return Value

The `killchar` function returns the user's current line kill character.

Name

leaveok – enable non-tracking cursor

Syntax

#include <cursesX.h>

int leaveok(win, bf)
WINDOW *win;
bool bf;

Description

This option allows the cursor to be left wherever the update happens to leave it. Normally, the cursor is left at the current location (y, x) of the window being refreshed. This routine is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

This option is initially disabled, and is not enabled until the value of `bf` is changed from `FALSE` to `TRUE`.

Return Value

The `leaveok` function returns OK on success and ERR on error.

longname(3cur)

Name

longname – return full terminal type name

Syntax

char *longname( )

Description

This routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to the `initscr` routine or the `newterm` routine.

The static area is overwritten by each call to `newterm` and is not restored by `set_term`. The value should be saved between calls to `newterm` if `longname` is going to be used with multiple terminals.

Return Value

The `longname` function returns a pointer to a verbose description of the current terminal on success and the null pointer on error.

See Also

initscr(3cur), newterm(3cur), set_term(3cur)

Name

meta – force the number of significant bits on input

Syntax

meta(win, bf)
WINDOW *win;
bool bf;

Description

This function forces the user's terminal to return 7 or 8 significant bits on input. To force 8 bits to be returned, invoke `meta` with `bf` as `TRUE`. To force 7 bits to be returned, invoke `meta` with `bf` as `FALSE`.

The window argument is always ignored, but it must still be a valid window to avoid compiler errors.

Name

move, wmove – move cursor in window

Syntax

move(y, x)

wmove(win, y, x)
WINDOW *win;
int y, x;

Description

The `move` routine moves the cursor associated with the default window to the given location (y, x), where `y` is the row, and `x` is the column. This routine does not move the physical cursor of the terminal until the `refresh` routine is called.

The `wmove` routine moves the cursor associated with the specified window to the given location (y, x). This does not move the physical cursor of the terminal until the `wrefresh` routine is called.

For both routines the position specified is relative to the upper left corner of the window, which is (0,0).

The routine `move` is a macro.

See Also

refresh(3cur), wrefresh(3cur)

Name

mvcur – low-level cursor movement

Syntax

mvcur(oldrow, oldcol, newrow, newcol)
int oldrow, oldcol, newrow, newcol;

Description

This function controls low-level cursor motion with optimization.

Name

mvwin – move window

Syntax

mvwin(win, y, x)
WINDOW *win;
int y, x;

Description

Move the window so that the upper left corner will be at position `(y, x)`. It is an error to move the window off the screen. If you try to do this the window is not moved.

Name

napms – sleep

Syntax

napms(ms)
int ms;

Description

This function causes the program to sleep for `ms` milliseconds. The number of milliseconds is limited to 1000.

newpad(3cur)

Name

newpad – create new pad

Syntax

#include <cursesX.h>

WINDOW *newpad(nlines, ncols)
int nlines, ncols;

Description

The `newpad` routine creates a new `pad` data structure. A pad differs from a window in that it is not restricted by the screen size, and it is not necessarily associated with a particular part of the screen. Pads can be used when large windows are needed. Only part of the pad will be on the screen at any one time.

Automatic refreshes of pads for example, from scrolling or echoing of input, do not occur.

You cannot call the `refresh` routine with a pad as an argument; use the routines `prefresh` or `pnoutrefresh` instead.

Note that these two routines require additional parameters to specify both the part of the pad to be displayed and the screen location for the display.

Return Value

On success the `newpad` function returns a pointer to the new WINDOW structure created. On failure the function returns a null pointer.

See Also

pnoutrefresh(3cur), prefresh(3cur), refresh(3cur)

Name

newterm – open new terminal

Syntax

#include <stdio.h>
#include <cursesX.h>

SCREEN *newterm(type, outfd, infd)
char *type;
FILE *outfd, *infd;

Description

Programs using more than one terminal should call the `newterm` routine for each terminal instead of `initscr`. The routine `newterm` should be called ONCE for each terminal.

The `newterm` routine returns a variable of type SCREEN * which should be saved as a reference to that terminal. There are three arguments. The first argument `type`, is the type of the terminal to be used in place of TERM. The second argument, `outfd`, is a file pointer for output to the terminal. The third argument, `infd`, is a file pointer for input from the terminal. The program must also call the `endwin` routine for each terminal, after each terminal has finished running a `curses` application.

Return Value

On success the `newterm` function returns a pointer to the new SCREEN structure created. On failure the function returns a null pointer.

See Also

endwin(3cur), initscr(3cur)

Name

newwin – create new window

Syntax

#include <cursesX.h>

WINDOW *newwin(nlines, ncols, begin_y, begin_x)
int nlines, ncols, begin_y, begin_x;

Description

The function `newwin` creates a new window with the number of lines, `nlines`, and columns, `ncols`. The upper left corner of the window is at line `begin_y`, column `begin_x`.

If either `nlines` or `ncols` is zero, they will be defaulted to LINES - `begin_y` and COLS - `begin_x`. A new full-screen window is created by calling `newwin`(0,0,0,0).

Return Value

On success the `newwin` function returns a pointer to the new WINDOW structure created. On failure the function returns a null pointer.

Name

nl, nonl – enable/disable newline control

Syntax

#include <cursesX.h>

int nl( )

int nonl( )

Description

The `nl` routine enables the newline control translations. When newline control is enabled, a newline is translated into a carriage return and a linefeed on output, and a return is translated into a newline on input. Initially, these translations do occur.

The `nonl` routine disables these translations, allowing the `curses` program to use the linefeed capability of the terminal, resulting in faster cursor motion. The `nl` routine is a macro.

Return Value

The `nl` and `nonl` functions return OK on success and ERR on error.

Name

nodelay – disable block during read

Syntax

#include <cursesX.h>

int nodelay(win, bf)
WINDOW *win;
bool bf;

Description

This option causes the `getch` routine to be a non-blocking call. If no input is ready, and `nodelay` is enabled, `getch` will return the integer `ERR`. If `nodelay` is disabled, `getch` will wait until input is ready.

Return Value

The `nodelay` function returns OK on success and ERR on error.

See Also

getch(3cur)

Name

overlay, overwrite – overlay windows

Syntax

#include <cursesX.h>

int overlay(srcwin, dstwin)
WINDOW *srcwin, *dstwin;

int overwrite(srcwin, dstwin)
WINDOW *srcwin, *dstwin;

Description

The `overlay` routine copies all the text from the source window `srcwin` on top of the destination window `dstwin`. The two windows are not required to be the same size. The copy starts at (0, 0) on both windows. The copy is non-destructive, so blanks are not copied.

The `overwrite` routine copies all of `srcwin` on top of `destwin`. The copy starts at (0, 0) on both windows. This is a destructive copy as blanks are copied.

Return Value

The `overlay` and `overwrite` functions return OK on success and ERR on error.

prefresh (3cur)

Name

prefresh, pnoutrefresh – refresh pad

Syntax

#include <cursesX.h>

int prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
WINDOW *pad;
int pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol;

int pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
WINDOW *pad;
int pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol;

Description

The `prefresh` routine copies the specified pad to the physical terminal screen. It takes account of what is already displayed on the screen to optimize cursor movement.

The `pnoutrefresh` routine copies the named pad to the virtual screen. It then compares the virtual screen with the physical screen and performs the actual update.

These routines are analogous to the routines `wrefresh` and `wnoutrefresh` except that pads, instead of windows, are involved. Additional parameters are also needed to indicate what part of the pad and screen are involved. The upper left corner of the part of the pad to be displayed is specified by `pminrow` and `pmincol`. The co-ordinates `sminrow, smincol, smaxrow,` and `smaxcol` specify the edges of the screen rectangle that will contain the selected part of the pad.

The lower right corner of the pad rectangle to be displayed is calculated from the screen co-ordinates. This ensures that the screen rectangle and the pad rectangle are the same size.

Both rectangles must be entirely contained within their respective structures.

Return Value

The `prefresh` and `pnoutrefresh` functions return OK on success and ERR on error.

See Also

wnoutrefresh(3cur), wrefresh(3cur)

Name

printw, mvprintw, mvwprintw, wprintw – formatted write to a window

Syntax

```
#include <cursesX.h>

int printw(fmt [, arg] ...)
char *fmt;

int wprintw(win, fmt [, arg] ...)
WINDOW *win;
char *fmt;

int mvprintw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;

int mvwprintw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

Description

The `printw` routine adds a string to the default window starting at the current cursor position. This routine causes the string that would normally be output by `printf` to be output by `addstr`.

The routine `wprintw` adds a string to the specified window starting at the current cursor position. This routine causes the string that would normally be output by `printf` to be output by `waddstr`.

The routine `mvprintw` adds a string to the default window starting at the specified cursor position. This routine causes the string that would normally be output by `printf` to be output by `addstr`.

The routine `mvwprintw` adds a string to the specified window starting at the specified cursor position. This routine causes the string that would normally be output by `printf` to be output by `waddstr`.

All these routines are analogous to `printf`. It is advisable to use the field width options of `printf` to avoid leaving unwanted characters on the screen from earlier calls.

Return Values

The `printw`, `mvprintw`, `mvwprintw`, and `wprintw` functions return OK on success and ERR on error.

See Also

addstr(3cur), waddstr(3cur), printf(3s)

Name

putp – pad and output a string

Syntax

putp(str)
char *str;

Description

The `putp` routine outputs the string `str` one character at a time. The routine `putchar` is used to control the output.

See Also

putchar(3s)

Name

raw, noraw – enable/disable raw mode

Syntax

int raw( )

int noraw( )

Description

The `raw` routine sets the terminal into RAW mode. RAW mode is similar to CBREAK mode, in that characters are immediately passed through to the user program as they are typed. In RAW mode, the interrupt, quit, suspend and flow control characters are passed through uninterpreted, and do not generate a signal.

The behavior of the BREAK key depends on the settings of bits that are not controlled by `curses`.

The `noraw` routine disables RAW mode.

Return Value

The `raw` and `noraw` functions return OK on success and ERR on error.

Name

refresh, wrefresh – refresh window

Syntax

#include <cursesX.h>

int refresh( )

int wrefresh(win)
WINDOW *win;

Description

The routine `wrefresh` copies the named window to the physical terminal screen, taking into account what is already there in order to optimize cursor movement.

The routine `refresh` does the same, using `stdscr` as a default screen.

These routines **must** be called to get any output on the terminal, as other routines only manipulate data structures.

Unless `leaveok` has been enabled, the physical cursor of the terminal is left at the location of the window's cursor. The routine `refresh` is a macro.

Return Value

The `refresh` and `wrefresh` functions return OK on success and ERR on error.

See Also

leaveok(3cur)

Name

resetty, savetty – restore/save terminal modes

Syntax

int resetty()

int savetty()

Description

The `savetty` routine saves the current state of the terminal modes in a buffer. The routine `resetty` restores the state of the terminal modes to what it was at the last call to `savetty`.

Return Value

The `resetty` and `savetty` functions return OK on success and ERR on error.

reset_prog_mode (3cur)

Name

reset_prog_mode, reset_shell_mode – restore terminal mode

Syntax

int reset_prog_mode( )

int reset_shell_mode( )

Description

The `reset_prog_mode` routine restores the terminal modes to those saved by the `def_prog_mode` routine.

The `reset_shell_mode` routine restores the terminal modes saved by the `def_shell_mode` routine.

These routines are called automatically by `endwin` and `doupdate` after an `endwin`. Normally these routines would not be called before `endwin`.

Return Value

The `reset_prog_mode` and `reset_shell_mode` functions return OK on success and ERR on error.

See Also

def_prog_mode(3cur), def_shell_mode(3cur), doupdate(3cur), endwin(3cur)

Name

restartterm – restart terminal for curses application

Syntax

restartterm(term, filenum, errret)
char *term;
int filenum;
int *errret;

Description

This function sets up the current terminal `term` after a save/restore of a `curses` application program. `restartterm` assumes that the windows and modes are the same for the restarted application as when memory was saved. It assumes that the terminal type and dependent settings, such as baudrate, may have changed. The routine `setupterm` is called to extract the terminal information from the `terminfo` database and set up the terminal.

See Also

setupterm(3cur), terminfo(5)

Name

scanw, mvscanw, mvwscanw, wscanw – formatted read from window

Syntax

```
#include <cursesX.h>

int scanw(fmt [, arg] ...)
char *fmt;

int wscanw(win, fmt [, arg] ...)
WINDOW *win;
char *fmt;

int mvscanw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;

int mvwscanw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

Description

These routines correspond to `scanf`. The function `scanw` reads input from the default window. The function `wscanw` reads input from the specified window. The function `mvscanw` moves the cursor to the specified position and then reads input from the default window. The function `mvwscanw` moves the cursor to the specified position and then reads input from the specified window.

For all the functions, the routine `wgetstr` is called to get a string from the window, and the resulting line is used as input for the scan. All character interpretation is carried out according to the `scanf` function rules.

Return Value

Upon successful completion, the `scanw`, `mvscanw`, `mvwscanw` and `wscanw` functions return the number of items successfully matched. On end-of-file, they return EOF. Otherwise they return ERR.

See Also

wgetstr(3cur), scanf(3s)

Name

scroll – scroll window

Syntax

#include <cursesX.h>

int scroll(win)
WINDOW *win;

Description

The window is scrolled up one line. This involves moving the lines in the window data structure.

You would not normally use this routine as the terminal scrolls automatically if `scrollok` is enabled. A typical case where `scroll` might be used is with a screen editor.

Return Value

The `scroll` function returns OK on success and ERR on error.

See Also

scrollok(3cur)

Name

scrollok – enable screen scrolling

Syntax

#include <cursesX.h>

int scrollok(*win, bf*)
WINDOW **win*;
bool *bf*;

Description

This option controls what happens when the cursor is moved off the edge of the specified window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled, (*bf* is FALSE) the cursor is left on the bottom line. If enabled, the window is scrolled up one line and then refreshed.

Return Value

The `scrollok` function returns OK on success and ERR on error.

Name

setscrreg, wsetscrreg – set scrolling region

Syntax

#include <cursesX.h>

int setscrreg(top, bot)
int top, bot;

int wsetscrreg(win, top, bot)
WINDOW *win;
int top, bot;

Description

The `setscrreg` routine sets the scrolling region for the default window.

The `wsetscrreg` routine sets the scrolling region for the named window. Use these routines to set a software scrolling region in a window.

For both routines, the line numbers of the top and bottom margins of the scrolling region are contained in `top` and `bot`. Line 0 is the top line of the window.

If this option and `scrollok` are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. Only the text of the window is scrolled.

Return Value

No return values are defined for these functions.

See Also

scrollok(3cur)

setupterm (3cur)

Name

setupterm – perform low level terminal setup

Syntax

setupterm(term, filenum, errret)
char *term;
int filenum;
int *errret;

Description

This function sets up the terminal from the `terminfo` database. The parameter `term` is the terminal type. If this parameter is set to `NULL` then the environment variable `TERM` will be used. The `filenum` parameter is an ULTRIX file descriptor, not a `stdio` pointer. It is used for all the output generated by `setupterm`.

The `terminfo` boolean, numeric and string values are stored in a structure of type `TERMINAL`.

After `setupterm` returns successfully the variable `cur_term` is initialized. This variable points to the `TERMINAL` structure. The `cur_term` pointer can be saved before calling `setupterm` again as further calls to `setupterm` allocate new space; the space pointed to by `cur_term` is not overwritten.

See Also

restartterm(3cur)

Name

set_term – switch between terminals

Syntax

#include <cursesX.h>

SCREEN *set_term(new)
SCREEN *new;

Description

This routine is used to switch between different terminals. The screen reference `new` becomes the new current terminal. The previous terminal screen reference is returned by the routine.

This is the only routine which manipulates SCREEN pointers; all the others change the current terminal only.

Return Value

The `set_term` function returns a pointer to the previous SCREEN structure on success and a null pointer on error.

Name

subwin – create subwindow

Syntax

#include <cursesX.h>

WINDOW *subwin(orig, nlines, ncols, begin_y, begin_x)
WINDOW *orig;
int nlines, ncols, begin_y, begin_x;

Description

This routine creates a new sub-window within a window. The dimensions of the sub-window are `nlines` lines and `ncols` columns. The sub-window is at position (`begin_y, begin_x`) on the screen. This position is relative to the screen, and not to the window `orig`.

The sub-window is made in the middle of the window `orig`, so that changes made to either window will affect both. When using this routine, it will often be necessary to call `touchwin` before calling `wrefresh`.

Return Value

On success the `subwin` function returns a pointer to the new WINDOW structure created. On failure the function returns a null pointer.

See Also

touchwin(3cur), wrefresh(3cur)

Name

tgetent, tgetnum, tgoto, tgetstr, tgetflag – emulate termcap for old programs

Syntax

```
int tgetent(bp, name)
char *bp, *name;

int tgetflag(id)
char *id;

tgetnum(id)
char *id;

tgetstr(id, area)
char *id, *area;

tgoto(cap, col, row)
char *cap;
int col, row;
```

Description

All these functions are included for compatibility with application programs that used the old `termcap` database.

Do not use these functions in new `curses` application programs.

Name

touchwin – touch window

Syntax

#include <cursesX.h>

int touchwin(win)
WINDOW *win;

Description

This routine discards all optimization information for the specified window and assumes that the entire window has been drawn on.

This is sometimes necessary when using overlapping windows, as a change to one window will affect the other window. The records of which lines have been changed may not be correct for the window which has not been changed directly.

Return Value

The `touchwin` function returns OK on success and ERR on error.

Name

tparm – instantiate a string

Syntax

char *tparm(str, p1, p2, ...)

Description

This function instantiates the string `str` with the parameters p1, p2, ... A pointer is returned which points to the result of `str` with the parameters applied.

Name

tputs – pad and output string

Syntax

```
tputs(str, count, putc)
register char *str;
int count;
int (*putc)();
```

Description

This function adds padding to the string `str` and outputs it. The string must be either a `terminfo` string variable or the return value from `tparm`, `tgetstr` or `tgoto`. The variable `count` is the number of lines affected; this is set to 1 if not applicable. The function `putc` is a `putchar` style routine. The characters are passed to `putc` one at a time.

See Also

putchar(3s), terminfo(5), tparm(3cur)

Name

traceon, traceoff – enable or disable debug trace output

Syntax

traceon()

traceoff()

Description

These functions turn the debugging trace output on and off when you use the debug version of the `curses` library `/usr/lib/libdcursesX.a`.

Name

typeahead – check for typeahead

Syntax

int typeahead(fd)
int fd;

Description

If `typeahead` is enabled, the `curses` program looks for typeahead input periodically while updating the screen. If input is found, the current update will be postponed until `refresh` or `doupdate` is called again. This allows faster response to commands typed in advance.

Normally, the input FILE pointer passed to the `newterm` routine, will be used to do this typeahead checking. If the routine `initscr` was called, the input FILE pointer is passed to `stdin`.

The `typeahead` routine specifies that the file descriptor `fd` is to be used to check for typeahead. If `fd` is –1, then typeahead is disabled.

Return Value

No return values are defined for this function.

See Also

doupdate(3cur), initscr(3cur), newterm(3cur), refresh(3cur)

Name

unctrl – convert character to printable form

Syntax

#include <cursesX.h>

char *unctrl(c)
chtype c;

Description

The `unctrl` routine expands the character `c` into a character string which is a printable representation of the character.

Control characters are displayed in the ^X notation. Printing characters are displayed normally. The `unctrl` routine is a macro, defined in the `unctrl.h` header file. This header file is included by the `cursesX.h` header file described in `intro`(3cur), so you do not have to include it again.

Return Value

The `unctrl` macro returns a string.

See Also

intro(3cur)

Name

vidattr, vidputs – output a string that sets terminal display

Syntax

vidattr(attrs)
vidputs(attrs, putc)

Description

The `vidattr` routine outputs a string that sets the video attributes `attrs` for the terminal. The characters in the string are passed one at a time to the routine `putchar`.

The `vidputs` routine is similar, except that the string characters are passed to the routine `putc`. Video attributes are described in *The Guide to X/Open Curses Screen-Handling*

See Also

putchar(3s)
Guide to X/Open Curses Screen-Handling

Name

wnoutrefresh, doupdate – do efficient refresh

Syntax

#include <cursesX.h>

int wnoutrefresh(win)
WINDOW *win;

int doupdate( )

Description

The `wnoutrefresh` routine updates screens more efficiently than using the `wrefresh` routine by itself. The `wnoutrefresh` routine copies the named window to a data structure referred to as the virtual screen (stdscr). The virtual screen contains what a program intends to display on the physical terminal screen. The routine `doupdate` compares the virtual screen to the physical screen and then does the actual update. These two routines allow multiple updates with more efficiency than `wrefresh`.

The routine `wrefresh` works by calling `wnoutrefresh`, and then calling `doupdate`. If a programmer wants to output several windows at once, a series of calls to `wrefresh` will result in alternating calls to `wnoutrefresh` and `doupdate`, causing several bursts of output to the screen. If `wnoutrefresh` is called first for each window, `doupdate` only needs to be called once, resulting in only one burst of output. This usually results in fewer total characters being transmitted and less CPU time used.

Return Value

The `doupdate` and `wnoutrefresh` functions return OK on success and ERR on error.

See Also

wrefresh(3cur)